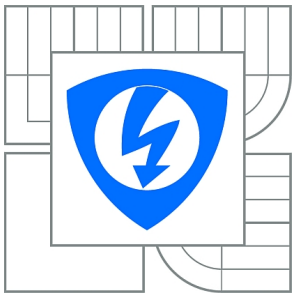
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF BIOMEDICAL ENGINEERING

ZASÍLÁNÍ SERVISNÍCH ZPRÁV V PROSTŘEDÍ WINDOWS

SERVICE MESSENGER FOR WINDOWS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

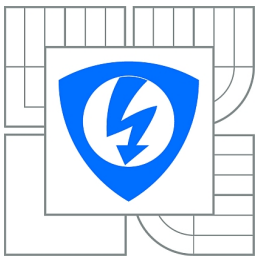
AUTOR PRÁCE — Ing. PETR POPELKA

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ SEKORA

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav biomedicínského inženýrství

Bakalářská práce

bakalářský studijní obor
Biomedicínská technika a bioinformatika

Student:	Ing. Petr Popelka	***ID:***	83332
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Zasílání servisních zpráv v prostředí Windows

POKYNY PRO VYPRACOVÁNÍ:

1) Seznamte se se sadou protokolů TCP/IP a prostudujte možnosti zasílání krátkých datagramů prostřednictvím sítě Ethernet. 2) Navrhněte aplikaci pro zasílání servisních zpráv pro platformu Windows. Aplikace musí umožňovat zaslání zpráv vybrané skupině příjemců (stanic), jejich archivaci a přijem. Zasílané zprávy ponesou informaci o identifikaci skupiny, o časové platnosti a samotný obsah. 3) Realizujte navrženou aplikaci. 5) Navrhněte a diskutujte možnou implementaci systému zasílání zpráv do informačního systému Oddělení klinické biochemie FN Brno. Správnost řešení otestujte.

DOPORUČENÁ LITERATURA:

[1] DOSTÁLEK, Libor, Marta VOHNOUTOVÁ a Miroslav KNOTEK. Velký průvodce infrastrukturou PKI a technologií elektronického podpisu. 2., aktualiz. vyd. Brno: Computer Press, 2009, 542 s. ISBN 978-80-251-2619-6.
[2] DRÁB, Martin. Jádro systému Windows. Brno: Computer Press, 2009, 542 s. ISBN 978-80-7222-823-2.

Termín zadání:	11.2.2013	***Termín odevzdání:***	31.5.2013

Vedoucí práce: Ing. Jiří Sekora
Konzultanti bakalářské práce:

prof. Ing. Ivo Provazník, Ph.D.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

V této práci je popsán proces komunikace v síti Ethernet založený na modelu TCP/IP. Na základě tohoto modelu byla navržena a realizována aplikace klient – server, která by mohla být využita pro zasílání servisních zpráv mezi skupinami lékařů a pracovištěm OKB ve FN Brno. Práce zmiňuje současné vybavení laboratoře a způsob komunikace jednotlivých zařízení s laboratorním informačním systémem pomocí protokolu ASTM. Tato komunikace pak může být využita k rozvoji dalších funkcí navržené aplikace.

KLÍČOVÁ SLOVA

TCP/IP, Ethernet, servisní zpráva, protokol, klient, server

ABSTRACT

This thesis describes the process of communication in Ethernet based on TCP/IP model. Client – server application was designed and realized using this model. This application can be used to send service messages between groups of doctors and workplace of OKB in FN Brno. This thesis also describes current laboratory equipment and communication between them laboratory information system using the protocol ASTM. This communication can be later used to develop other features of the designed application.

KEYWORDS

TCP/IP, Ethernet, service message, protocol, client, server

POPELKA, P. *Zasílání servisních zpráv v prostředí Windows*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. 2013. 41s. Vedoucí bakalářské práce Ing. Jiří Sekora.

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma „Zasílání servisních zpráv v prostředí Windows“ samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne …… …….

Podpis autora

Poděkování

Děkuji vedoucímu své bakalářské práce Ing. Jiřímu Sekorovi, paní RNDr. Miroslavě Beňovské Ph.D a panu Ing. Petru Kopeckému za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne …… …….

Podpis autora

Obsah

1 Úvod 6
2 Síťový model TCP/IP 7
2.1 Vrstva síťového rozhraní 8
2.2 Síťová vrstva 8
2.3 Transportní vrstva 8
2.4 Aplikační vrstva 9
3 Ethernet 10
4 OKB ve FN Brno 11
4.1 Systém MPA 12
4.2 Systém COBAS 6000 13
4.3 Systém COBAS 8000 13
5 Softwarové vybavení 15
5.1 Protokol ASTM 15
5.1.1 Nízkoúrovňový protokol 15
5.1.2 Vysokoúrovňový protokol 16
6 Návrh aplikace 17
6.1 Funkce serveru 17
6.2 Funkce klienta 18
7 Programová část 20
7.1 Server 21
7.1.1 Funkce ServerThread 24
7.2 Klient 28
7.2.1 Funkce ClientThread 31
8 Závěr 34
Seznam literatury 35
Seznam zkratek a symbolů 36
Seznam příloh 36

Seznam obrázků

Obrázek 2.1: Síťový model TCP/IP 7
Obrázek 2.2: Zapouzdřování dat 7
Obrázek 4.1: Počítačová síť OKB 11
Obrázek 4.2: Systém MPA ve FN Brno (topologie I) 12
Obrázek 6.1: Databáze klientů a fronta soketů 18
Obrázek 6.2: Formát přenášené zprávy 19
Obrázek 7.1: Komunikační schéma Klient-Server 20
Obrázek 7.2: Obsluha příchozích spojení 23
Obrázek 7.3: Vývojový diagram podmínek identifikující zprávu 25
Obrázek 7.4: Soubor settings.txt 28
Obrázek 7.5: Klient01 32
Obrázek 7.6: Klient02 32
Obrázek 7.7: Server 33
Obrázek 7.8: Logovací souborlog.txt 33
Obrázek A.1: Vývojový diagram serveru (1. část) 37
Obrázek A.2: Vývojový diagram serveru (2. část) 38
Obrázek A.3: Vývojový diagram klienta 39
Obrázek B.4: Spuštěná aplikace Cinderella server 40
Obrázek B.5: Aplikace Cinderella klient 41

1 Úvod

Někdy může být při komunikaci po síti výhodnější komunikovat přímo s celou skupinou uživatelů než s jednotlivými uživateli. Tato bakalářská práce se věnuje možností návrhu aplikace pro zasílání servisních zpráv v prostředí Windows skupině uživatelů a také možností zda a jak by se tato aplikace dala integrovat do informačního systému na oddělení klinické biochemie ve fakultní nemocnici v Brně. V první části projektu se zaměříme na popis modelu TCP/IP abychom si dokázali představit, co se děje na jednotlivých vrstvách nejpoužívanějšího komunikačního modelu nejen v síti internet. Následně si představíme, jak probíhá komunikace a přístup ke komunikačnímu médiu na hojně používané technologii, především pro sítě LAN, Ethernet.

Protože by měla být navržená aplikace nasazena na OKB ve FN Brno pro komunikaci mezi laboranty a lékaři, seznámíme se krátce s analyzátory používanými na tomto oddělení a také a to především jak probíhá komunikace s analyzátory a laboratorním informačním systémem na protokolu ASTM.

Následně bude představen návrh na realizaci takovéto aplikace. Dále budou prodiskutovány nejen možnosti integrace navržené aplikace do laboratorního informačního systému, ale také možnost zpracování informací zasílaných laboratorními analyzátory protokolem ASTM navrženou aplikací nebude-li integrace možná.

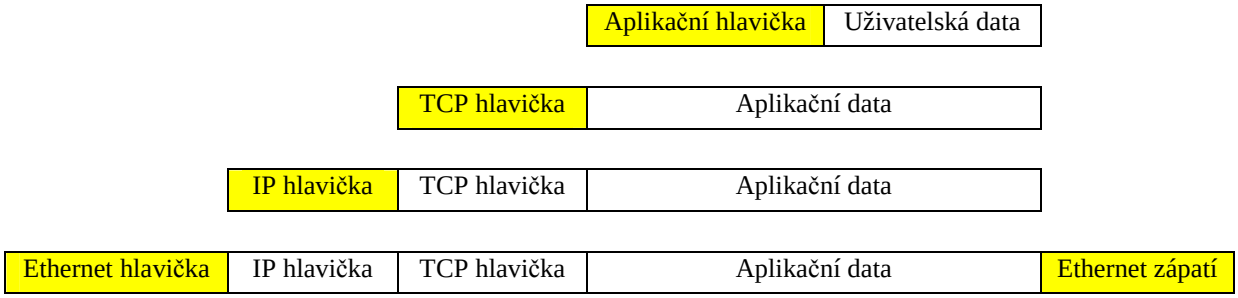
Aplikace pro zasílání zpráv bude realizována v jazyce C++ a bude rozdělena do dvou samostatných částí. Jednou částí je server starající se o zpracování všech zpráv odeslaných klienty tak také o jejich logování. Druhou částí je klient, který pouze zprávy odesílá nebo přijímá zprávy jemu určené.

2 Síťový model TCP/IP

V dnešní době se pro komunikaci v síti používá nejvíce modelu TCP/IP. Na rozdíl od sedmi vrstvého modelu ISO/OSI [2] se jedná pouze o čtyř vrstvý model zajišťující přenos dat od odesílatele k příjemci. Je tvořen vrstvou aplikační, transportní, síťovou a vrstvou síťového rozhraní [1]. Model TPC/IP je znázorněn na Obrázek 2.1.

Obrázek 2.1: Síťový model TCP/IP

Při odesílání dat se provádí tzv. zapouzdřování od nejvyšší vrstvy směrem k nejnižší. Postup je následující: aplikace vezme data, která chce odeslat, přidá k nim svou aplikační hlavičku a tento nový datagram předá nižší vrstvě (transportní vrstvě). Tato vrstva vezme takto nově vzniklý datagram, opět přidá svoji hlavičku a postoupí následující nižší vrstvě [1]. To se opakuje, než data přijdou k nejnižší vrstvě, která pak data odešle svému příjemci jako posloupnost bitů. Při přijetí dat koncovou stanicí se tento děj opakuje avšak v opačném pořadí. Na Obrázek 2.2 můžeme vidět zjednodušený příklad zapouzdřování dat v modelu TCP/IP.



Obrázek 2.2: Zapouzdřování dat

2.1 Vrstva síťového rozhraní

Není v modelu TCP/IP blíže specifikována a je závislá na použité přenosové technologii. Tato nejnižší vrstva poskytuje přístup k vlastnímu fyzickému médiu a řízení datového spoje. Někdy je tato vrstva nazývána jako *Ethernetová* z důvodu nejčastějšího nasazení Ethernetu jako přenosové technologie [1]. Technologii ethernet se budeme věnovat v následující kapitole. Dalšími přenosovými technologiemi jako je ISDN, ATM, FDDI, Token Ring, … se zabývat nebudeme.

2.2 Síťová vrstva

Účelem této vrstvy je aby se odesílaná data dostaly co nejdříve svému příjemci, což také v důsledku znamená, že nezajišťuje spolehlivost doručení přenášených dat. Spolehlivost si tedy musí řešit vyšší vrstvy popřípadě samotné aplikace. O směřování datagramů (paketů) od zdroje k cílovému zařízení přes jednu nebo více IP sítí se stará protokol IP (*Internet Protocol*). V paketu jsou obsažena řídící data a data uživatelská. Řídící data obsahují informace k doručení paketu jako je, adresa zdroje a cíle, informace o pořadí, kontrolní součty, atp. Každý uzel v síti a každé koncové zařízení je identifikováno IP adresou, která musí být v této síti unikátní.

IP adresa protokolu IPv4 je 32 bitové číslo, které se nejčastěji zapisuje jako čtyři desítkové čísla oddělené tečkami např.: 192.168.0.1. Tato adresa bývá dále doplněna o tzv. masku podsítě. Ta určuje kolik bitů adresy je určeno k identifikaci podsítě a kolik pro identifikaci konkrétních stanic v dané podsíti.

2.3 Transportní vrstva

Další vrstvou TCP/IP je transportní vrstva (*Transport Layer*), někdy také TCP vrstva, jelikož je nejčastěji realizována právě TCP protokolem. Hlavním úkolem této vrstvy je zajistit přenos mezi dvěma koncovými účastníky, kterými jsou v případě TCP/IP přímo aplikační programy [1]. Podle jejich nároků a požadavků může transportní vrstva regulovat tok dat oběma směry, zajišťovat spolehlivost přenosu a také měnit nespojovaný charakter přenosu (v síťové vrstvě) na spojovaný [1].

Dalším významným protokolem operujícím na transportní vrstvě je protokol UDP (User Datagram Protocol), který na rozdíl od TCP nezajišťuje mj. spolehlivost přenosu. UDP protokol je výhodné použít tam, kde je kladen důraz na co nejmenší zpoždění a kde ztráta určitého počtu datagramů ještě neovlivní kvalitu služby. Jsou to především služby

zprostředkovávající přenos dat v reálném čase jako např. streamování videa, IP telefonie atp. Protokol UDP umí také v rámci dané podsítě odesílat data všesměrově.

Jednotlivé aplikace jsou na transportní vrstvě identifikovány tzv. číslem portu, které nabývá hodnot od 1 do 65535, z nichž je prvních 1024 portů rezervováno systémem.

2.4 Aplikační vrstva

Nejvyšší vrstvou TCP/IP je pak vrstva aplikační (Application Layer). Jejími entitami jsou jednotlivé aplikační programy, které na rozdíl od referenčního modelu ISO/OSI komunikují přímo s transportní vrstvou. Případné prezentační a relační služby, které v modelu ISO/OSI zajišťují samostatné vrstvy, si zde musí jednotlivé aplikace v případě potřeby realizovat samy [1].

3 Ethernet

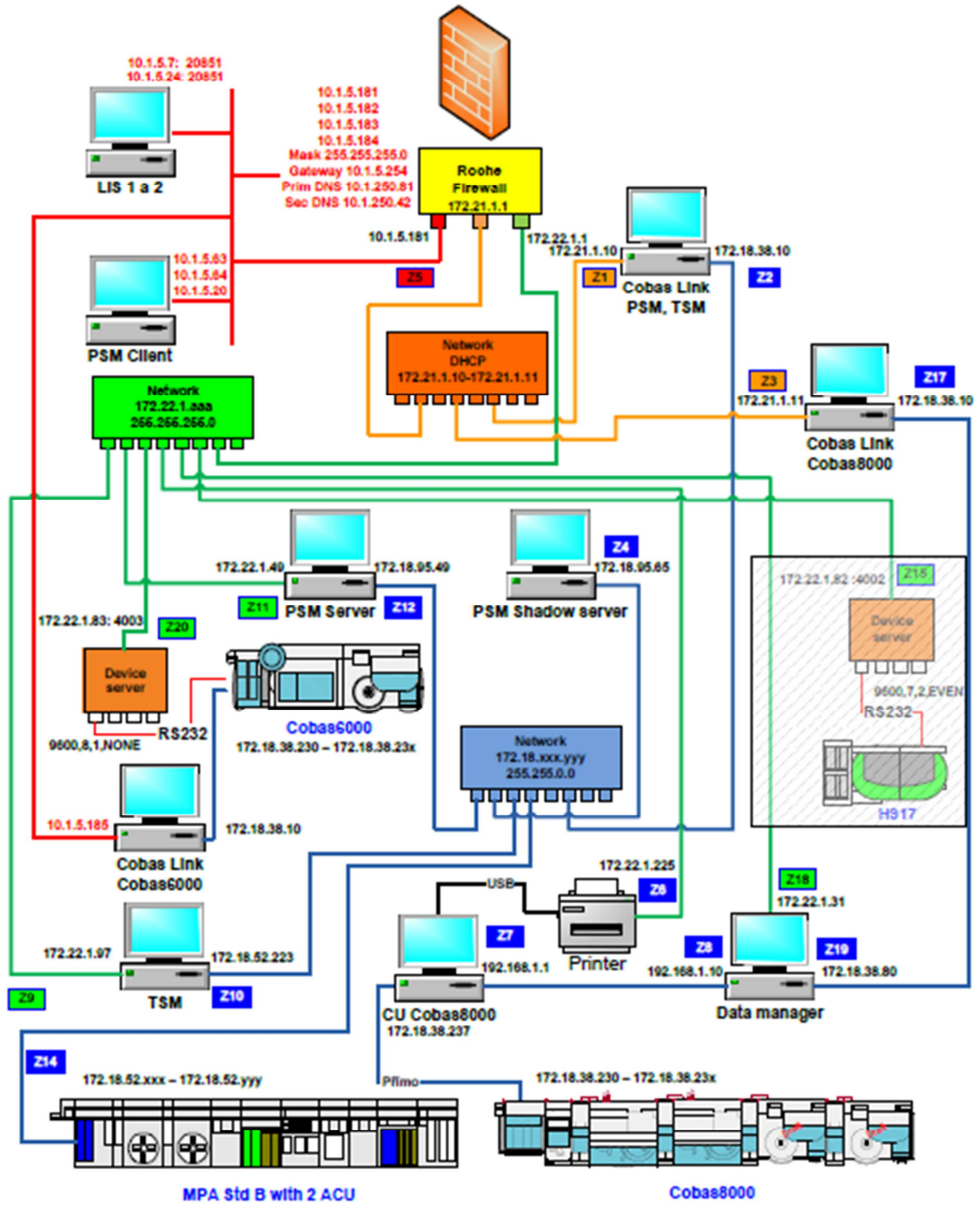
Ethernet je v současné době nejvíce používanou síťovou technologií. Zpočátku využíval ethernet jako přenosového média koaxiálního kabelu. Postupně časem se začalo používat kroucených dvojlinek, optických kabelů a v neposlední řadě také bezdrátového přenosu.

Přístup k přenosovému médiu je zajištěn pomocí jednoduché metody CSMA/CD. Jedná se o metodu s vícenásobným přístupem, naslouchánim na nosné s detekcí kolizí. Stanice vysílající data nejprve naslouchá, zda je médium volné. Pokud není, čeká na jeho uvolnění. Dojde-li k uvolnění média, stanice začne vysílat data a současně naslouchá na přenosovém médiu, zdali nezachytí vysílání od jiného zdroje. Dojde-li ke kolizi, stanice vysílání zastaví, vygeneruje tzv. signál JAM. Všechny uzly účastnící-se kolize zastaví vysílání, vyčkají náhodnou dobu a poté se znovu pokusí o vysílání. Nevýhodou tohoto řešení je, že s přibývajícím počtem uzlů narůstá i počet kolizí a tím klesá propustnost sítě. Soubor uzlů, jejichž činnost může vést ke kolizi, se nazývá kolizní doména [3].

Původně se jako přenosového média využívalo koaxiálního kabelu v zapojení sdílené sběrnice s přenosovou rychlostí 10Mb/s. Postupně s vývojem nových technologií se začalo používat kroucené dvojlinky a nově také optických vláken. Dnes je nejčastěji k realizaci sítě LAN nasazována kroucená dvojlinka s využitím přepínačů jako aktivních prvků kde se již nevyužívá sdílené médium. Použitím přepínačů docílíme odstranění kolizní domény tedy spojení typu bod-bod. Přenosová rychlost této technologie je 100Mb/s a je označována jako Fast Ethernet (100Base-TX). Na této síťové technologii je také postavena síť ve FN Brno.

4 OKB ve FN Brno

Na OKB ve FN Brno se pro automatickou analýzu vzorků používá několik modulárních systému systémů od firmy Roche, které jsou vzájemně propojeny počítačovou sítí ethernet. Jedná se o systém MPA, Cobas 6000 a nejnovější Cobas 8000. Počítačová síť laboratoře OKB je znázorněna na obrázku Obrázek 4.1 [4].



Obrázek 4.1: Počítačová síť OKB

V práci bude popsán především analyzátor Cobas 8000, který je napojen na systém MPA.

4.1 Systém MPA

Jedná se o stavebnicový systém zajišťující pre- a postanalytické funkce. Je to online automat přímo napojený na analyzátor Modular PPPE, tedy systém se třemi moduly pro klinickou chemii a jedním pro imunoanalýzu [4]. Typickými výhodami tohoto systému jsou především, odstranění možnosti chyb v distribuci vzorků, snížení možnosti kontaminace biologickým materiálem při manipulaci se vzorkem, sledování pohybu vzorku, úspora pracovní síly a zlepšení časové odezvy. Systém může být uspořádán do tvaru I, L nebo U. Konfigurace instalovaná ve FN Brno je uspořádána do tvaru I viz Obrázek 4.2 [4] a skládá se z těchto částí:

- vstupní modul
- dvě centrifugy
- odzátkovací modul
- aliquotační modul
- modul lepící čárové kódy na zkumavky s aliquoty
- zátkovací modul
- třídič
- výstupní modul

Obrázek 4.2: Systém MPA ve FN Brno (topologie I)

Rychlost systému je závislá na tom, kterými moduly vzorek prochází. Je-li vzorek centrifugován je rychlost přibližně 400 vzorků za hodinu.

4.2 Systém COBAS 6000

Cobas 6000 je modulární analytický systém pro středně velké klinické laboratoře [5]. Systém využívá tří měřících principů:

- ISE (iontově selektivní elektrody) – pro měření elektrolytů Na, K, Cl
- Fotometrickou jednotku – pro klasickou fotometrii a turbidimetrii
- Heterogenní modul – pro elektrochemiluminiscenční měření

Tyto měřící principy se využívají také u systému Cobas 8000 a tak bude jejich podrobnější popis uveden v následující kapitole.

4.3 Systém COBAS 8000

Cobas 8000 je modulární, plně automatický, počítačově řízený systém pro fotochemické a imunochemické analýzy, určené pro kvantitativní a kvalitativní in-vitro stanovení široké palety testů [6]. Na oddělení OKB ve FN Brno je instalovaný systém Cobas 8000 skládající se z následujících částí:

- **Core jednotky** – je základem každého systému. Obsahuje vstupní jednotku, výstupní jednotku a obslužný systém pro dopravu pětimístných stojánků k jednotlivým modulům. Dále obsahuje statimový vstup pro vkládání stojánků s urgentními vzorky, nebo při napojení na Modular MPA pro on-line vstup vzorků. Core jednotka je schopna obsluhovat až 4 analytické moduly a až 2 ISE jednotky [5].

- **Dva ISE moduly** – slouží k měření koncentrace Na, K, Cl iontů v tělních tekutinách za pomocí iontově selektivních elektrod. Každý modul obsahuje dvě samostatné ISE (iontově selektivní elektrody jednotky. Výkon každé jednotky ISE je až 300 vzorků za hodinu.

- **Dva moduly c701** – umožňují in-vitro stanovení řady analytů jako cholesterol, transferin, glukóza atp. v biologických materiálech pomocí fotometrické metody [6]. Každý modul můžeme rozdělit na tyto dílčí části – vzorková a reagenční část a prostor reakčního kruhu. Vzorková část obsahuje dva pipetory, které pipetují vzorky ze zkumavek do reakčních kyvet. V reagenční části se nachází dva reagenční kruhy, ve kterých jsou uloženy reagencie a které obsluhují čtyři pipetory. Tyto reagenční pipetory přenáší reagencie do kyvet uložených v reakčním kruhu. Kapacita kruhů je 70 souprav. V prostoru reakčního kruhu je také umístěn fotometrický systém a šest ultrazvukových míchadel. Výkon modulu c701 je až 2000 testů za hodinu [6].

- **Dva moduly e602** – moduly pro heterogenní imunoanalýzu využívající ke stanovení látek o velmi nízkých koncentracích (vitamíny, léky, tumorových markerů, …) elektrochemiluminiscence [9]. Modul má dvě měřící komory a jeho výkon je až 170 testů za hodinu. Reagenční kruh má 25 pozic. Prakticky je tedy možné provádět 18 – 22 různých testů současně [5]. Pro jeden test je možno použít jen jednu nebo obě měřící komory podle frekvence požadavků na příslušný test, čímž je možno snížit náklady na kalibraci.

- **Zásobníky ke každému z modulů –** různé druhy reagencií a destilovaná voda na proplachování kyvet

5 Softwarové vybavení

Software je nepostradatelným vybavením pro řízení celého systému, kontrolu údajů i stavu a zpracování veškerých dat. Cobas link využívá internet pro denně aktualizovanou databázi aplikací, kalibrátorů a kontrol, možnost dálkové správy atd. [5]. Další softwarovou součástí je Data Manager, který slouží pro propojení s LIS (laboratorní informační systém) a také koordinuje přenos dat v reálném čase mezi analyzátorem a PSM (proces system manager). PSM je middleware, který zprostředkovává komunikaci mezi LIS (Laboratorní informační systém), analyzátory a MPA [6]. Ke komunikaci se na aplikační vrstvě používá protokolu ASTM [7].

5.1 Protokol ASTM

Mezi nejrozšířenější protokol ke komunikaci mezi laboratorními přístroji v medicíně patří protokol ASTM. Protokol ASTM specifikuje obsah zprávy pro přenos informací mezi klinickými zařízeními a počítačovým systémem. Rozlišujeme jej na nízkoúrovňový a vysokoúrovňový. Nízkoúrovňová část protokolu popisuje jak přidat řídící informace do reálných dat popsaných vysokoúrovňovou částí protokolu [7].

5.1.1 Nízkoúrovňový protokol

1. Fáze navázání spojení

   Odesílatel oznamuje příjemci, že má připravená data k odeslání znakem <ENQ>. Je-li příjemce připraven data přijmout, odpoví odesílateli znakem <ACK>. Tím fáze navázání spojení končí a přechází se na fázi přenosu dat. Není-li příjemce na příjem dat připraven, odešle znak <NAK>. Odesílatel pak čeká nejméně 10 sekund, znovu odesílá znak <ENQ> a celý proces se opakuje [7].

2. Fáze přenosu

   Během přenosové fáze vysílač odesílá zprávy přijímači, dokud nejsou všechny odeslány. Zprávy jsou odesílány v rámcích, jejichž maximální délka je 247 znaků včetně hlavičky. Zprávy delší než 240 znaků jsou rozděleny na dva a více rámců. Existují dva typy rámců a to střední rámec a ukončovací [7]. Střední rámec končí znaky <ETB>, kontrolní součet, <CR><LF>. Koncový rámec končí znaky <ETX>, kontrolní součet, <CR><LF>.

5.1.2 Vysokoúrovňový protokol

Vysokoúrovňový protokol ASTM definuje různé typy zpráv a seznam polí pro každou z nich. Pole jsou oddělena oddělovači. Které oddělovače se budou používat, následuje bezprostředně za hlavičkou zprávy [7]. U protokolu ASTM se používá těchto oddělovačů:

| - oddělovač pole

\ - opakovací oddělovač

^ - oddělovač složek

& - návratový znak

V textových datových polích jsou povoleny pouze ASCII znaky 32 – 126 a 128 – 254 vyjma výše zmíněných oddělovačů. Za sledování zprávy neobsahuje-li nepovolené znaky je zodpovědný vysílač. Délka polí i celé zprávy je proměnné délky a tek z tohoto důvodu vyžadována u přijímače vyrovnávací paměť. Protokol ASTM definuje následující typy zpráv [7]:

H – Hlavičková zpráva

P – Zpráva s údaji o pacientovy

O – Zpráva objednávky testu

R – Zpráva s výsledkem testu

C – Zpráva s komentářem

D – Zpráva s dotazem

L – Ukončovací zpráva

Popis jednotlivých polí všech zpráv můžeme nalézt např. v [7] strana B-9 – B-24. Níže je zobrazen příklad, jak taková komunikace na vysoké úrovni může probíhat.

H|\^&|||HOST-NAME|||||||P|1|20110516082356
P|1||1234
O|1|1234||^^^NA\^^^K\^^^CL|R|20110406100000|20110406100000||||A|||||||
|||||||
O|2|1234||^^^GLU|R|20110406100000|20110406100000||||A||||||||||||
L|1|N

6 Návrh aplikace

Aplikace bude navržena jako samostatně fungující program a nebude integrována do informačního systému OKB jelikož se jedná o uzavřený systém firmy Roche, která v žádném případě neumožňuje implementaci aplikací třetích stran do jejich softwaru. Nicméně se zde naskytuje možnost využít komunikace protokolem ASTM mezi LIS a analyzátory a tak implementovat některé funkce do naší aplikace jako je zobrazování statimových vzorků, které jsou na pracovišti déle než 60 minut a vzorků z vitální indikace, které jsou na pracovišti déle než 30 minut nebo zobrazování posledních výsledků kontrol všech rutinních metod (zeleně) - budou-li odchýleny o víc než 2 SD žlutě a více než 3 SD červeně. Tyto požadavky budou ještě dále diskutovány, přestože nebudou zahrnuty do tohoto projektu.

Aplikace je rozdělena do dvou částí, serverové a klientské. Spojení bude na transportní vrstvě realizováno protokolem TCP. Klient se připojí k serveru pomocí tzv. soketu. Soket je jakýsi pomyslný přípojný bod, přes který může aplikace odesílat data do sítě nebo je naopak přijímat [8]. Spojení mezi dvěma aplikacemi běžícími na různých stanicích si můžeme představit jako fiktivní propojovací kabel vedoucí z jedné zásuvky do druhé, kterým proudí data [8].

6.1 Funkce serveru

Serverová aplikace se bude starat o přijímání zpráv od klientů, tyto zprávy pak bude přeposílat pouze klientům patřících do dané skupiny uvedené ve zprávě. Aktuálně připojení klienti k serveru budou uloženi v databázi na serveru. Dále se server bude starat o ukládání všech příchozích zpráv do souboru pro případné reporty a statistiky. Vývojový diagram serverové aplikace znázorněn v příloze A na obrázcích Obrázek A.1 a Obrázek A.2.

Server nejprve načte ze souboru nastavení IP adresy, na které bude přijímat požadavky. Nastavení portu je standardně 15687. Poté přejde do fáze vytvoření a pojmenování soketu. Proběhne-li vše v pořádku, vytvoří se fronta, do které se postupně zařazují požadavky klientů. Po vyzvednutí požadavku z fronty se ze soketu vytvoří nový soket tzv. potomek, na němž bude probíhat obsluha konkrétního klienta. Definovány jsou tři typy zpráv:

1. Zpráva REGIST – obdrží-li server tento typ zprávy, začne se procházet databáze klientů. Je-li v ní klient nalezen, může to znamenat, že předchozí spojení se serverem nebylo ukončeno korektně anebo se hlásí nový uživatel se stejným uživatelským jménem. V obou případech dojde ke smazání původního záznamu a vytvoří se nový záznam na jiné pozici v databázi. Databáze obsahuje ID klienta, jméno klienta a skupinu. ID klienta je spárováno s frontou otevřených soketů viz. Obrázek 6.1: Databáze klientů a fronta soketůObrázek 6.1.

2. Zpráva UNREGIST – po obdržení této zprávy dojde k odeslání zprávy „ExItCoDe1937" a zrušení soketu, na němž byl klient obsluhován (usmrcení potomka). Následně se projde databáze a odstraní se v ní klientův záznam.

3. Libovolná zpráva – je jakákoli jiná zpráva zasílaná klientem určité skupině. Po přijetí zprávy, server načte systémový čas a společně s uživatelským jménem, skupinou (příjemců) a zprávou ji uloží na následující volný řádek do logovacího souboru. Pak se identifikuje skupina příjemců. Jedná-li se o skupinu „00“ odešle se zpráva všem aktivním klientům (např. zpráva: Celý den nepoteče voda), jinak se odesílá pouze těm, kteří mají v databázi na místě skupiny stejný identifikátor jako je obsažený ve zprávě.

Tento cyklus se neustále opakuje do doby, než program ukončíme.

ID	1	2	3	…
Soket ID:	Soket 1	Soket 2	Soket 3	…

ID	Jméno	Skupina
1		
2		
4		
5		
6		
…		

Obrázek 6.1: Databáze klientů a fronta soketů

6.2 Funkce klienta

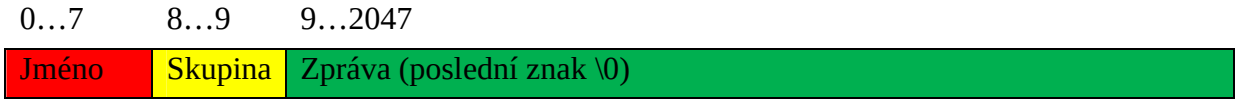
Klientská aplikace může běžet na kterémkoli PC v síti. Její vývojový diagram je znázorněn na obrázku Obrázek A.3 v příloze A. Po spuštění aplikace dojde nejprve k nastavení parametrů ze souboru tj. uživatelské jméno klienta (text o 8 znacích a číslech), skupina (libovolné 2 ASCII znaky) a IP adresa serveru. Poté dojde k vytvoření a nastavení TCP soketu. Bylo-li spojení se serverem navázáno v pořádku, klient ihned odešle zprávu REGIST. Následně čeká buď na odeslání zprávy, nebo přijetí nové zprávy. Při přijetí zprávy mohou nastat tyto dva případy.

1. Standardní zpráva – zpráva se klientovi zobrazí v zobrazovacím poli
2. Zpráva „ExItCoDe1937“ – dojde k uzavření soketu a zobrazení informace, že se klient odpojil o serveru

Při odesílání zprávy se nejprve určí, které skupině se má zpráva odeslat. Pak se zpráva odešle a je-li serverem potvrzeno přijetí, celý děj se opakuje. Napíšeme-li v konzoli klienta slovo

„exit“ a potvrdíme, odešle se serveru zpráva UNREGIST. Ten pak potvrdí zprávou „ExItCoDe1937“, že byl klient odpojen, uzavře se soket a program se ukončí.

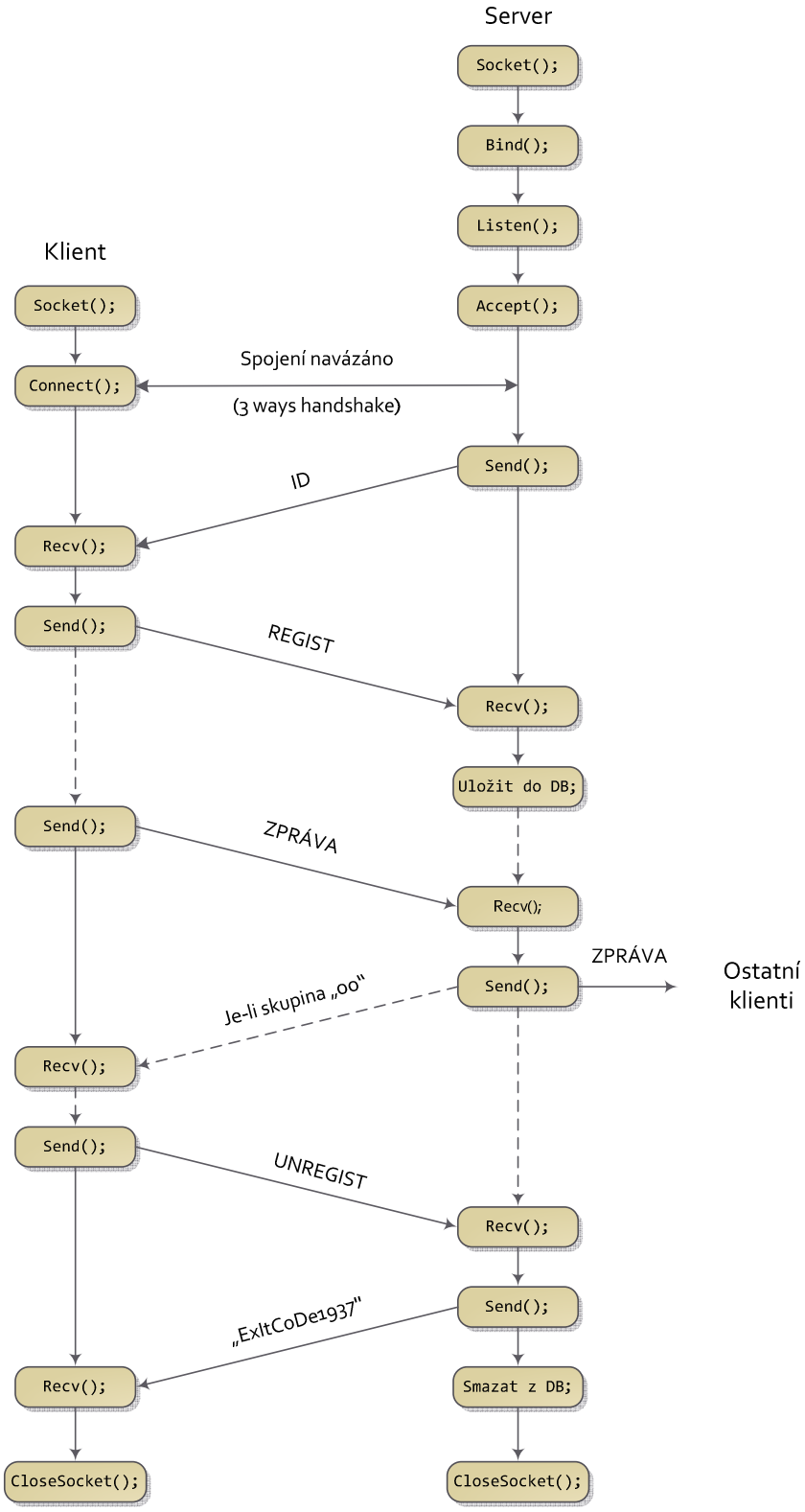
Formát celé zprávy je naznačen na Obrázek 6.2. Hodnoty udávají počet znaků (bytů).



Obrázek 6.2: Formát přenášené zprávy

7 Programová část

Obě části jsou naprogramovány v jazyce C++ jako konzolové aplikace. Na obrázku Obrázek 7.1 je naznačeno komunikační schéma mezi klientem a serverem.



Obrázek 7.1: Komunikační schéma Klient-Server

7.1 Server

Pro chod serverové aplikace je důležité použít správné hlavičkové soubory, které obsahují některé důležité funkce. Jsou to:

stdio.h – terminálový vstup a výstup. Slouží především pro vypisování informací na obrazovku, čtení hodnot zadávaných klávesnicí, …

Winsock2.h – obsahuje funkce pro práci se sokety, které implementovány v knihovním souboru ws2_32.dll. Tuto knihovnu je třeba předat linkeru pomocí příkazu

```
#pragma comment(lib, "Ws2_32.lib")
```

Windows.h – hlavičkový soubor obsahující deklarace všech funkcí Windows API, všechna běžná makra a všechny datové typy používané různými funkcemi a subsystémy.

iostream – informace o tom jak má jazyk komunikovat s okolím, např. přijímat údaje od uživatele nebo vypisovat text.

string – umožňuje použití datového typu *string*

ctime – obsahuje definice funkcí k získání a práci s časem

fstream – funkce pro práci se soubory (čtení / zápis)

Funkce initWinsock – tato funkce se stará o inicializaci soketů. Návratová hodnota je typu integer. Zdrojový kód funkce je zobrazen níže.

```
int initWinSock()
{
	int RetVal = 0;
	WSAData wsaData;
	WORD DllVersion = MAKEWORD(2,1);
	RetVal = WSAStartup(DllVersion, &wsaData);

	return RetVal;
}
```

Volání funkce v programu je následující:

```
int RetVal = 0;
RetVal = initWinSock();

if(RetVal != 0)
```

```
{
     MessageBoxA(NULL, "Inicializace Winsock selhala", "Error",
     MB_OK | MB_ICONERROR);
     exit(1);
}
```

Pokud je návratová hodnota různá od 0 pak došlo při inicializaci soketů k chybě, zobrazí se chybová hláška a program je ukončen.

Důležité pro správnou funkci je aby server věděl na jaké IP adrese a portu má server komunikovat. Port je pevně nastaven na hodnotu 15687. IP adresa se načítá z konfiguračního souboru settings.txt do proměnné SA. Další možností je načítat IP adresu přímo ze systému to však s sebou nese úskalí, vyskytují-li se na stanici více než dva síťové adaptéry.

Server po většinu času naslouchá, čeká na požadavek od klienta. V okamžiku, kdy se klient pokusí připojit, server přijme spojení, čímž vytvoří nový soket a pomocí něj komunikuje s klientem. Nejprve je tedy nutné vytvoření soketu a svázání jej s určitou adresou. To provedeme voláním funkce **bind** jejíž parametry jsou:

- Soket – se kterým se má operace provést
- Adresa – ukazatel na strukturu SOCKADDR
- Velikost struktury SOCKADDR

Volání funkce bind vypadá následovně. Funkce listen specifikuje maximální počet čekajících připojení (max. hodnota klientů připojených k serveru, v našem případě 255).

```
bind(sListen, (SOCKADDR*)&addr, sizeof(addr));
listen(sListen, 255);
```

Na obrázku Obrázek 7.2 je zobrazen vývojový diagram obsluhy přicházejících spojení od klientů na server.

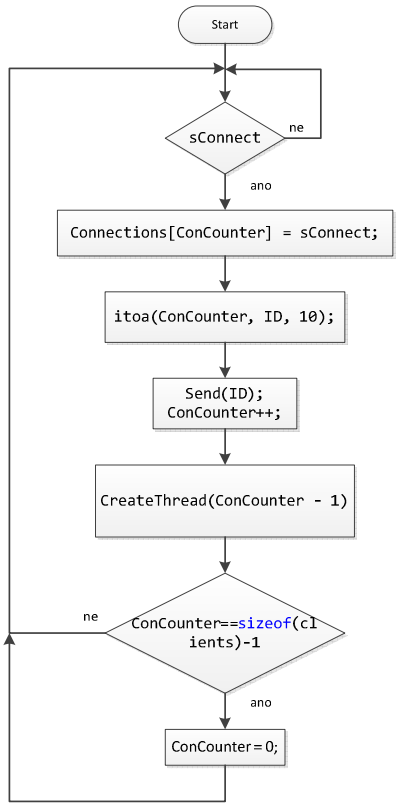
Proces probíhá v nekonečné smyčce. Přijde-li požadavek na spojení, to znamená, že volání funkce

```
sConnect = accept(sListen, (SOCKADDR*)&addr, &addrlen);
```

není 0, uloží se soket, na kterém je spojení navázáno do proměnné *Connections* na pozici *ConCounter*. *ConCounter* je počitadlo aktuálních připojení a zvyšuje se o 1 při každém nově příchozím spojení. Poté se hodnota *ConCounter* převede z datového typu **integer** na **char** a překopíruje do proměnné *ID*. Následně se *ID* odešle klientovi a *ConCounter* se zvýší o 1. Na závěr se pro aktuálního klienta vytvoří vlákno s parametrem *ConCounter-1* (ID klienta), na kterém pak probíhá samostatná komunikace mezi ním a serverem.

```
CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) ServerThread,
(LPVOID)(ConCounter - 1), NULL, NULL);
```

Podmínka na konci cyklu má následující funkci. Maximální počet obsluhovaných klientů je v krajním případě 255. Program však počítá s tím, že tato situace nikdy nenastane a počet klientů bude vždy << 255. To znamená, že neustále se připojujícím klientům se zvyšuje hodnota ID (*ConCounter*) až do hodnoty 255. Dojde-li k vyčerpání tj. *ConCounter* je roven 255, hodnota se vynuluje a začíná se od začátku. Tohle sice není ideální řešení, ovšem v našem případě je plně dostačující.



Obrázek 7.2: Obsluha příchozích spojení

7.1.1 Funkce ServerThread

Funkce ServerThread je stěžejní částí programu a proto jí je věnována samostatná podkapitola.

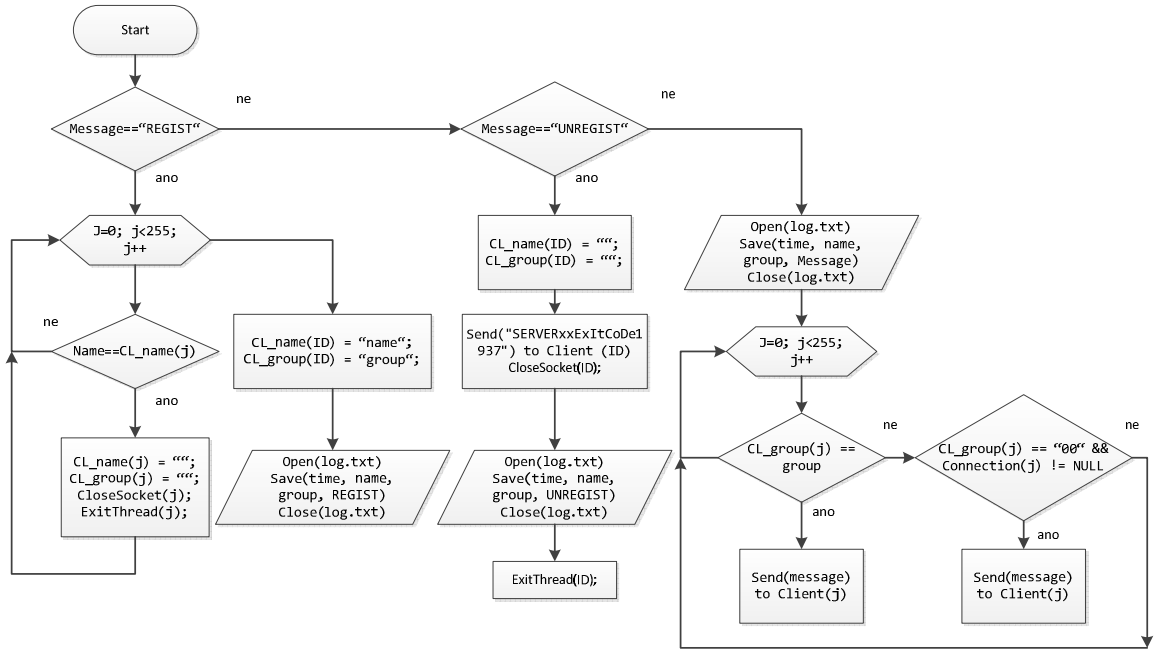
Jak již bylo zmíněno výše, funkce je volána pro každého klienta samostatně a probíhá v ní veškerá komunikace mezi klientem a serverem. Každé vlákno je identifikováno unikátním identifikátorem ID respektive hodnotou *ConCounter-1*. Ve vlákně běží další nekonečná smyčka, která se ukončí pouze tehdy, je-li ukončeno samostatné vlákno. Ve smyčce se nejprve detekuje příchozí zpráva funkcí

```
if(recv(Connections[ID], Recv, 2048, NULL))
```

Je-li zpráva přijata, zkopíruje se její obsah do struktury *sbuffer* do proměnné *Message*. Následně je zapotřebí rozlišit hlavičku zprávy (jméno, skupina) a samostatnou zprávu. O to se stará sled tří po sobě jdoucích smyček. Hodnoty jméno, skupina a zpráva se ukládají dvakrát. Jednou do proměnné typu **char** a jednou do proměnné typu **string**. Kód je následující.

```
for (int i=0; i<8; i++)
{
     name1[i] = sbuffer.Message[i];
}
name1[8]='\0';
string nameStr(name1);

for (int i=0; i<2; i++)
{
     group1[i] = sbuffer.Message[i+8];
}
group1[2] = '\0';
string groupStr(group1);

for (int i=0; i<2037; i++)
{
     message1[i] = sbuffer.Message[i+10];
                }
string messageStr(message1);
```

První smyčka ukládá 8 znaků jména, druhá 2 znaky skupiny a třetí zbývající znaky jako zprávu. Máme-li potřebné hodnoty takto uloženy, následuje sada podmínek zjišťujících, jaký typ zprávy byl přijat. Připomeňme si, že mohou nastat tři případy a to zpráva REGIST, UNREGIST a libovolná zpráva. Na následujícím obrázku Obrázek 7.3 je naznačeno podrobné řešení pomocí vývojového diagramu následovaného zdrojovým kódem s podrobným vysvětlením.



Obrázek 7.3: Vývojový diagram podmínek identifikující zprávu

1. Podmínka – zpráva „REGIST“

```
if(strcmp(message1,"REGIST")==0)
{
	for(int j=0; j < 255; j++)
	{
		if(strcmp(name1, clients[j].CL_name.c_str())==0)
		{
			clients[j].CL_name = "";
			clients[j].CL_group = "";
			closesocket(Connections[j]);
			ExitThread(j);
		}
	}

	clients[ID].CL_name = nameStr;
	clients[ID].CL_group = groupStr;

	myfile.open ("log.txt", ios::app);
```

```
        myfile << currentDateTime() <<"  " << nameStr <<"  " <<
        groupStr <<"   REGIST" << "\n";
        myfile.close();

    }
```

První podmínka testuje, přišla-li registrační zpráva „REGIST“ od nově připojeného klienta. Je-li tato podmínka splněna, prochází se databáze všech již registrovaných klientů a porovnává se jméno nově se registrujícího klienta se jménem na aktuální pozici v databázi. Při zjištění shody je záznam z databáze vymazán, uzavře se soket a ukončí se vlákno. Poté se uloží nový záznam do databáze na aktuální volnou pozici, která je dána parametrem *ID* a také do logovacího souboru spolu s aktuálním časem.

2. Podmínka – zpráva „UNREGIST“

```
else if(strcmp(message1,"UNREGIST")==0)
{
        clients[ID].CL_name = "";
        clients[ID].CL_group = "";

        strcpy(Send, "SERVER01xxExItCoDe1937\0");
        send(Connections[ID], Send, sizeof(Send), NULL);
        closesocket(Connections[ID]);

        myfile.open ("log.txt", ios::app);

        myfile << currentDateTime() <<"  " << nameStr <<"  " <<
        groupStr <<"   UNREGIST" << "\n";
        myfile.close();

        ExitThread(ID);
}
```

Přijde-li zpráva „UNREGIST“, pak se v následující podmínce provedou následující úkony. Smaže se záznam v databázi, do proměnné *Send* se zkopíruje zpráva „SERVERxxExItCoDe1937“ ukončená znakem \0 a ta se odešle zpět klientovi jako potvrzení o zrušení registrace. Část zprávy „SERVER01xx“ má pouze doplňující charakter aby byl zachován formát zprávy. Následuje uzavření soketu, uložení informace do logu s aktuálním časem a ukončení vlákna.

3. Podmínka – jakákoli jiná zpráva

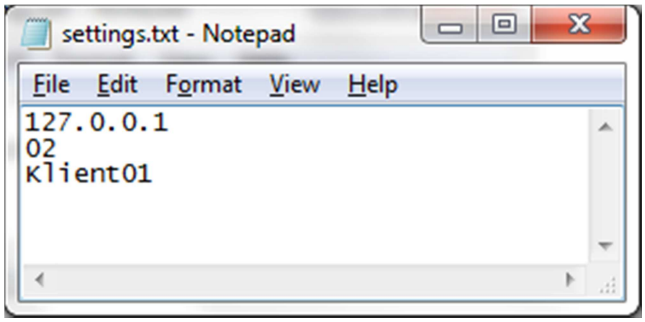
```
else
{
      myfile.open ("log.txt", ios::app);

      myfile << currentDateTime() <<"  " << nameStr <<"  " <<
      groupStr <<"  " << messageStr << "\n";
      myfile.close();

      for(int j=0; j < 255; j++)

      {
            char pom[3];
            strcpy(pom, clients[j].CL_group.c_str());

            if(groupStr==pom)
                  {
                  memcpy(Send, &sbuffer, sizeof(Buffer));
                  send(Connections[j], Send, sizeof(Buffer),
                  NULL);
            }
            else if(groupStr=="00" && Connections[j] != NULL)
            {
                  memcpy(Send, &sbuffer, sizeof(Buffer));
                  send(Connections[j], Send, sizeof(Buffer),
                  NULL);
            }
      }
      ZeroMemory(Recv, 2048);
}
```

Nenastal-li ani jeden ze dvou případů výše, musela být přijata zpráva nesoucí informaci pro další klienty. Tato zpráva se nejprve spolu s aktuálním časem, jménem a skupinou které zpráva náleží, uloží do logovacího souboru. Následně se začne procházet databáze klientů a porovnávat skupina obsažená v příchozí zprávě se skupinou v databázi u každého klienta. Je-li nalezena shoda, zpráva se odešle. Není-li tak otestuje, se zda není v příchozí zprávě uvedena skupina „00“ a je-li tohle splněno, zpráva se rovněž odešle ovšem pouze za současně platící podmínky, že pro daného klienta existuje soket. Nakonec se vynuluje celá paměť pro příchozí zprávy a celý cyklus se od přijetí další zprávy opakuje.

7.2 Klient

Klientská aplikace využívá ke svému chodu stejné hlavičkové soubory a knihovny jako server.

Po spuštění klientské aplikace se nejprve načítá nastavení ze souboru settings.txt. Ukázka souboru settings.txt je na obrázku Obrázek 7.4.



Obrázek 7.4: Soubor settings.txt

Zdrojový kód zobrazen níže načítá ve smyčce postupně všechny tři řádky a ukládá je do samostatných proměnných.

```
myfile.open ("settings.txt");

if (myfile.is_open())
{
      int k=0;
      while ( myfile.good())
      {
            getline(myfile, line[k]);
            k++;
      }
      myfile.close();
}else cout << "Chyba pri nacitani nastaveni";

client_name = line[2];
GRP = line[1];
grp = GRP;
string serverAddr = line[0];
```

IP adresa serveru uložená v proměnné *serverAddr* nemůže být typu **string,** protože funkce inet_addr vyžaduje proměnnou datového typu **char***. Je tedy nutné proměnnou *serverAddr* přetypovat na tento datový typ. To uděláme následujícím způsobem.

```
char *SA = new char[serverAddr.size() + 1];
copy(serverAddr.begin(), serverAddr.end(), SA);
SA[serverAddr.size()] = '\0';
```

Vytvoříme novou proměnnou typu **char*** o velikosti proměnné *serverAddr* + 1. Do ní pak po jednotlivých znacích zkopírujeme obsah *serverAddr* a na poslední místo vložím ukončovací znak \0.

Máme-li vše potřebné načteno, zobrazí se nám aktuální nastavení na obrazovce a začne se s inicializací soketů stejně jako v případě serverové aplikace. Proběhne-li inicializace v pořádku, vytvoříme soket.

```
sConnect = socket(AF_INET, SOCK_STREAM, NULL);
```

Parametr AF_INET značí, že se jedná o rodinu protokolů IPv4, SOCK_STREAM znamená, že jde o spojovaný typ přenosu pomocí protokolu TCP a třetí parametr není povinný, tudíž NULL.

Máme vytvořený soket, na kterém bude komunikace probíhat. Dále však ještě musíme naplnit strukturu *addr* typu SOCKADDR_IN. Ta obsahuje následující položky.

```
addr.sin_addr.s_addr = inet_addr(SA);
addr.sin_family = AF_INET;
addr.sin_port = htons(15687);
```

IP adresu, na kterou bude soket navázán (adresa serveru), rodinu protokolů (IPv4) a port.

Nyní máme vše připraveno pro navázání spojení se serverem. To provedeme příkazem

```
RetVal = connect(sConnect, (SOCKADDR*)&addr, sizeof(addr));
```

Vstupní parametry funkce `connect` jsou: soket, na kterém má komunikace probíhat, adresa serveru a její velikost. Funkce `connect` vrací hodnotu 0 v případě úspěšného navázání spojení jinak libovolnou hodnotu různou od 0. Po úspěšném navázání spojení, čeká klient od serveru zprávu vygenerovaným ID.

```
recv(sConnect, cID, 64, NULL);
```

Je-li ID doručeno, zobrazí se uživateli na obrazovce a vytvoří se samostatné vlákno *ClientThread*, které se bude starat o přijímání zpráv ze serveru viz podkapitola 7.2.1. Poté se pomocí níže uvedených příkazů vytvoří a odešle registrační zpráva „REGIST“.

```
packet = client_name;
packet.append(GRP);
packet.append("REGIST");

strcpy(buffer, packet.c_str());
buffer[2047] ='\0';

send(sConnect, buffer, 2048, NULL);
```

Do proměnné *packet* se uloží uživatelské jméno, doplňujícím příkazem *append* se připojí skupina a nakonec zpráva „REGIST“. Poté se obsah proměnné *packet* překopíruje znak po znaku do *bufferu* a na konec doplní ukončovací znak \0. Obsah *bufferu* je pak odeslán na server.

Samotné odesílání zpráv vytvořených klientem probíhá v nekonečné smyčce. Vždy je nejprve nutné zadat identifikátor skupiny, které chceme zprávu odeslat, a teprve pak napsat samotnou zprávu. Po potvrzení se začne zpráva testovat, zda neobsahuje klíčové slovo „exit“ nebo nepovolené zprávy „REGIST“ a „UNREGIST“. Sled příkazů je obdobný jako při odesílání registrační zprávy pouze s tím rozdílem, že v případě jedná-li se o zprávu „exit“ připojí se do *packetu* „UNREGIST“ a zpráva se odešle. V případě napíše-li uživatel „REGIST“ nebo „UNREGIST“, zobrazí se uživateli chybové hlášení a nic se neodesílá. Jde-li o libovolnou

jinou zprávu neodpovídající ani jednomu kritériu výše, odešle se zpráva zvolené skupině příjemců.

7.2.1 Funkce ClientThread

Funkce ClientThread je podobná funkci ServerThread s tím rozdílem, ClientThread existuje pouze jako jediná instance, kdežto instancí ServerThread může existovat až 255 identifikovaných právě parametrem ID. Její počet závisí na počtu aktivních klientů.

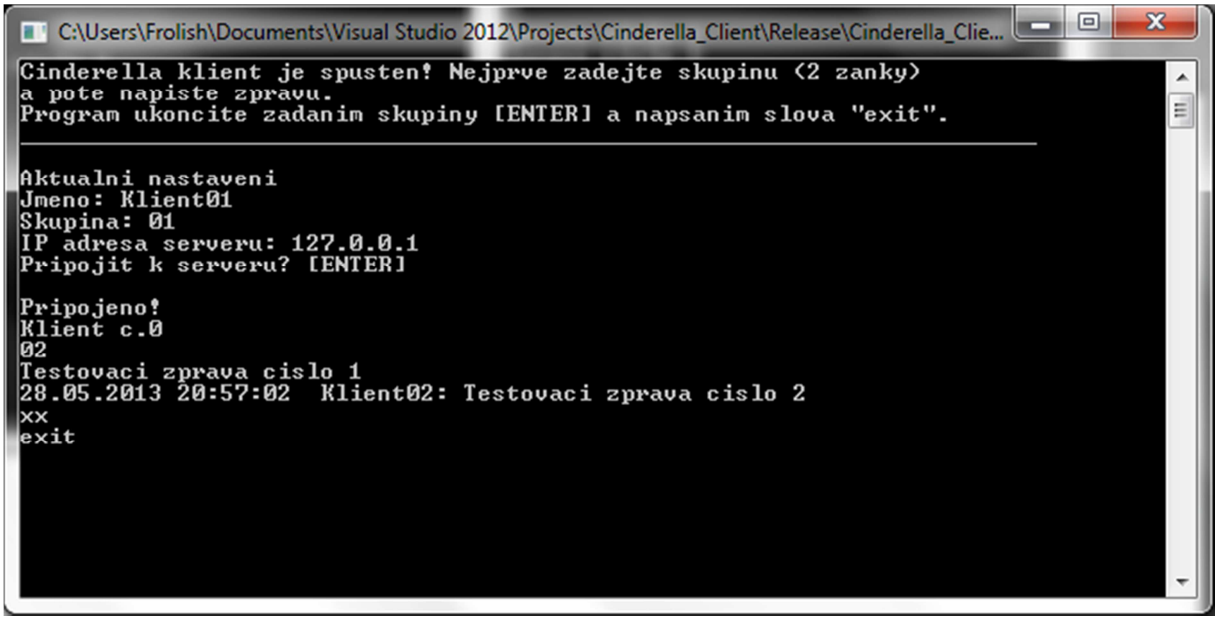
Po přijetí zprávy se provede stejně jako u serveru sada cyklů k rozlišení uživatelského jména, skupiny a samostatné zprávy. Po nich následuje podmínka, zjišťující zda nejde o zprávu „ExItCoDe1937“ (Pozn.: Zprávu „ExItCoDe1937“ zasílá server jako potvrzení o odpojení a zrušení registrace klienta). Je-li tato podmínka splněna, uzavře se soket, vyčistí se knihovna soketů Winsock 2, ukončí se vlákno ClientThread a následně celý program viz níže zdrojový kód.

```
if(strcmp(message1,"ExItCoDe1937")==0)
{
     cout << "Klient odpojen!" << endl;
     closesocket(sConnect);
     WSACleanup();
     ExitThread(ClientThread());
     getchar();
     return 0;
}
else
{
     cout << currentDateTime() << "  " << nameStr << ": " <<
     messageStr << endl;
}
```

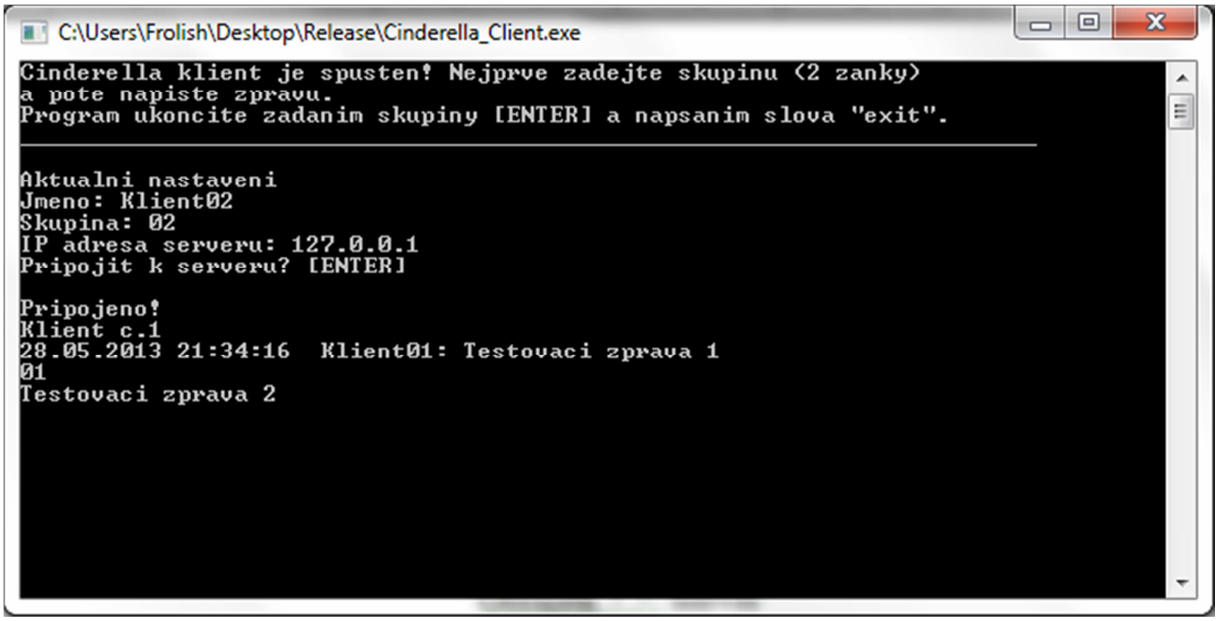
Není-li podmínka splněna, zobrazí se zpráva na obrazovce a celý cyklus se od přijetí další zprávy opakuje.

Na obrázcích Obrázek 7.5, Obrázek 7.6, Obrázek 7.7 je uveden reálný příklad komunikace mezi serverem a dvěma klienty. Průběh je následující. Nejprve je spuštěn server a následně první a druhý klient. Každý klient obdrží od serveru své číslo (ID). Poté zasílá první klient testovací zprávu číslo 1 skupině „02“, ve které se nachází druhý klient. Ten zprávu přijme a zobrazí. Pak druhý klient zasílá testovací zprávu číslo 2 skupině „01“, ve které se naopak

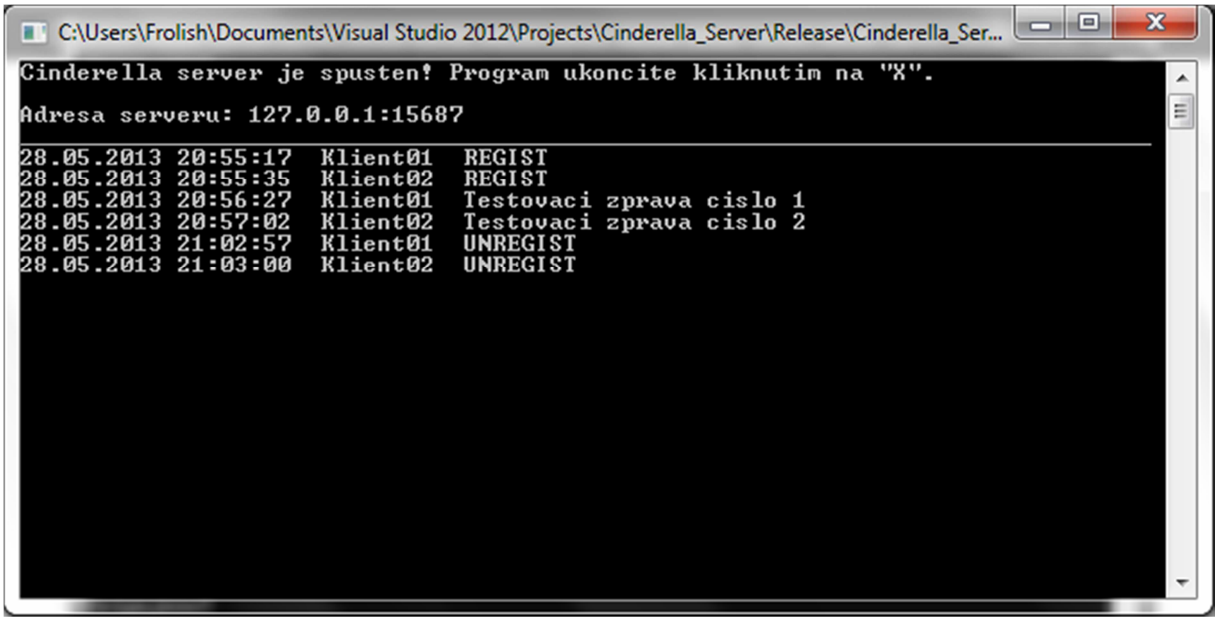
nachází první klient, který ji rovně přijímá a zobrazuje. Nakonec se oba klienti od serveru odhlašují.



Obrázek 7.5: Klient01

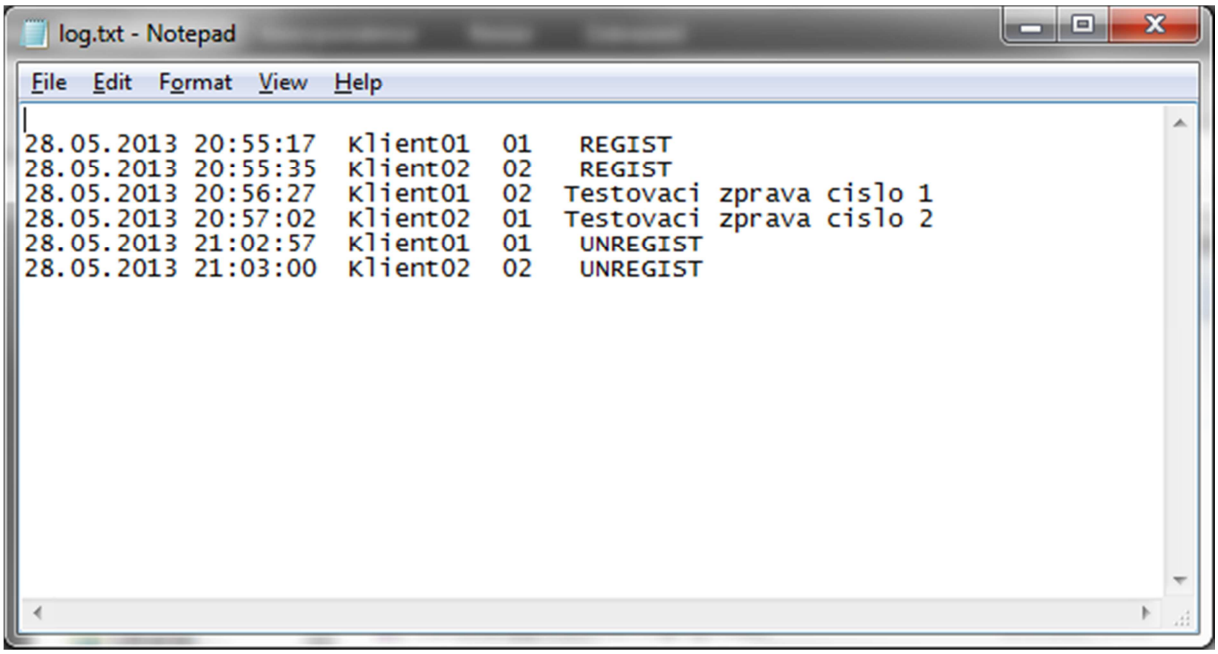


Obrázek 7.6: Klient02



Obrázek 7.7: Server

Výpis logovacího souboru vypadá následovně.



Obrázek 7.8: Logovací souborlog.txt

8 Závěr

Cílem této práce bylo navrhnout a realizovat aplikaci pro zasílání zpráv sítí Ethernet v prostředí Windows. K tomu bylo nutno se seznámit s touto přenosovou technologií a síťovým modelem TCP/IP. Vysvětlili jsme si o co se která vrstva modelu TCP/IP stará a popsali jsme si některé vlastnosti hlavních protokolů, kterými jsou TCP, UDP a IP.

Dále jsme se seznámili s analyzátory firmy Roche, které jsou instalovány v laboratoři na OKB ve FN Brno. Ukázali jsme si, jak jsou analyzátory zapojeny do sítě a jak komunikují s laboratorním informačním systémem.

Protože firma Roche neumožňuje integraci aplikací třetích stran přímo do jejich systému, byla navržena samostatně fungující aplikace, která funguje jako server – klient na protokolu TCP. Aplikace funguje tak, že zasílá servisní zprávy konkrétní skupině uživatelů. To je výhodné, zejména pokud chceme velmi rychle informovat větší skupinu uživatelů nebo potřebuje-li lékař co nejrychleji sdělit výsledky testů. Samotná aplikace byla realizována v jazyce C++. Dále je nutno zmínit, že program spousty nevýhod a nedodělků než komerčně používané komunikační programy jako např. Office Communicator 2010. Mezi hlavní nevýhody patří především nutnost pamatovat si, který uživatel patří do jaké skupiny, nutnost zadávat přesný počet znaků pro volbu skupiny atd. I přes tyto nevýhody je program pro naše účely použitelný a otevřený kód nám umožňuje program dále opravovat a rozvíjet o funkce, které běžné komunikátory nenabízí. Jde zejména o zpracování zpráv přenášených protokolem ASTM, jímž komunikují biochemické analyzátory COBAS s laboratorním informačním systémem v nemocnici v Brně na OKB. V budoucnu se tedy bude rozvoj této aplikace ubírat tímto směrem.

Seznam literatury

[1] PUŽMANOVÁ, R. *TCP/IP v kostce*. 2. vydání. KOPP, 2010. ISBN 978-80-7232 388-3

[2] PETERKA, J. EArchiv.cz. *Sedm vrstev ISO/OSI* [online]. 2011 [cit. 2012-12-12]. Dostupné z: http://www.earchiv.cz/a96/a625k150.php3

[3] ODVÁRKA, P. Svět sítí. *Ethernet* [online]. 2000 [cit. 2012-12-12]. Dostupné z: http://www.svetsiti.cz/clanek.asp?cid=Ethernet-1992000

[4] BEŇOVSKÁ, M., KOPECKÝ P. *Modular Preanalytics v FN Brno* [online]. 2008 [cit.2012-12-12]. Dostupné z: http://www.roche-diagnostics.cz/download/la/0208/Modular.pdf

[5] Roche Diagnostics. *Cobas 6000 modulární analytický systém* [online]. 2009 [cit. 2012-12-12]. Dostupné z: http://www.roche-diagnostics.cz/produkty/klinickelaboratore/cobas6000_popis.aspx

[6] Roche Diagnostics. *Cobas 8000 modulární analyzátor*. Uživatelská příručka 02-01, Leden 2010. Verze 2.0

[7] Roche Diagnostics. *PSM – Process System Manager – Host Interface Manual*. 2012. Version 2.03.01

[8] PIRKL, J. *Síťové programování pod Windows a programování Internetu*. KOPP, 2002. ISBN 80-7232-145-5.

[9] FRANČÍK, V. *Elecsys® 2010 v r. 2010 aneb z historie elektrochemiluminiscence a podíl Čecha na geniálním objevu*. 2010. Dostupné z: http://www.roche-diagnostics.cz/download/la/0210/Elecsys.pdf

Seznam zkratek a symbolů

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

IP – Internet Protokol

LAN – Local Area Network

OKB – Oddělení klinické biochemie

FN – Fakultní nemocnice

LIS – Laboratorní informační systém

PSM – Process system manažer

DHCP – Dynamic Host Configuration Protocol

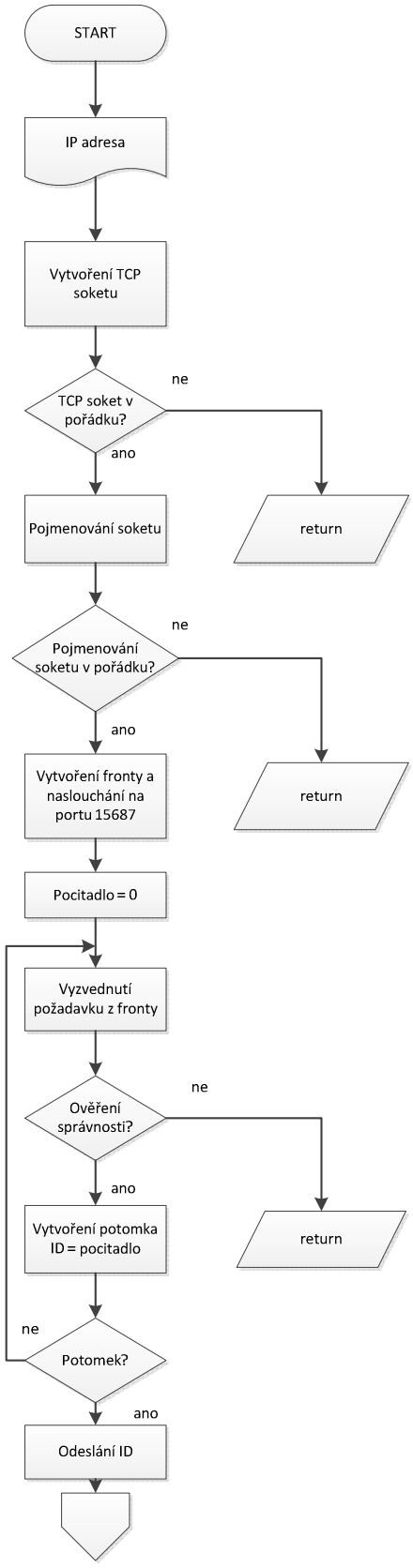
SD – směrodatná odchylka

Seznam příloh

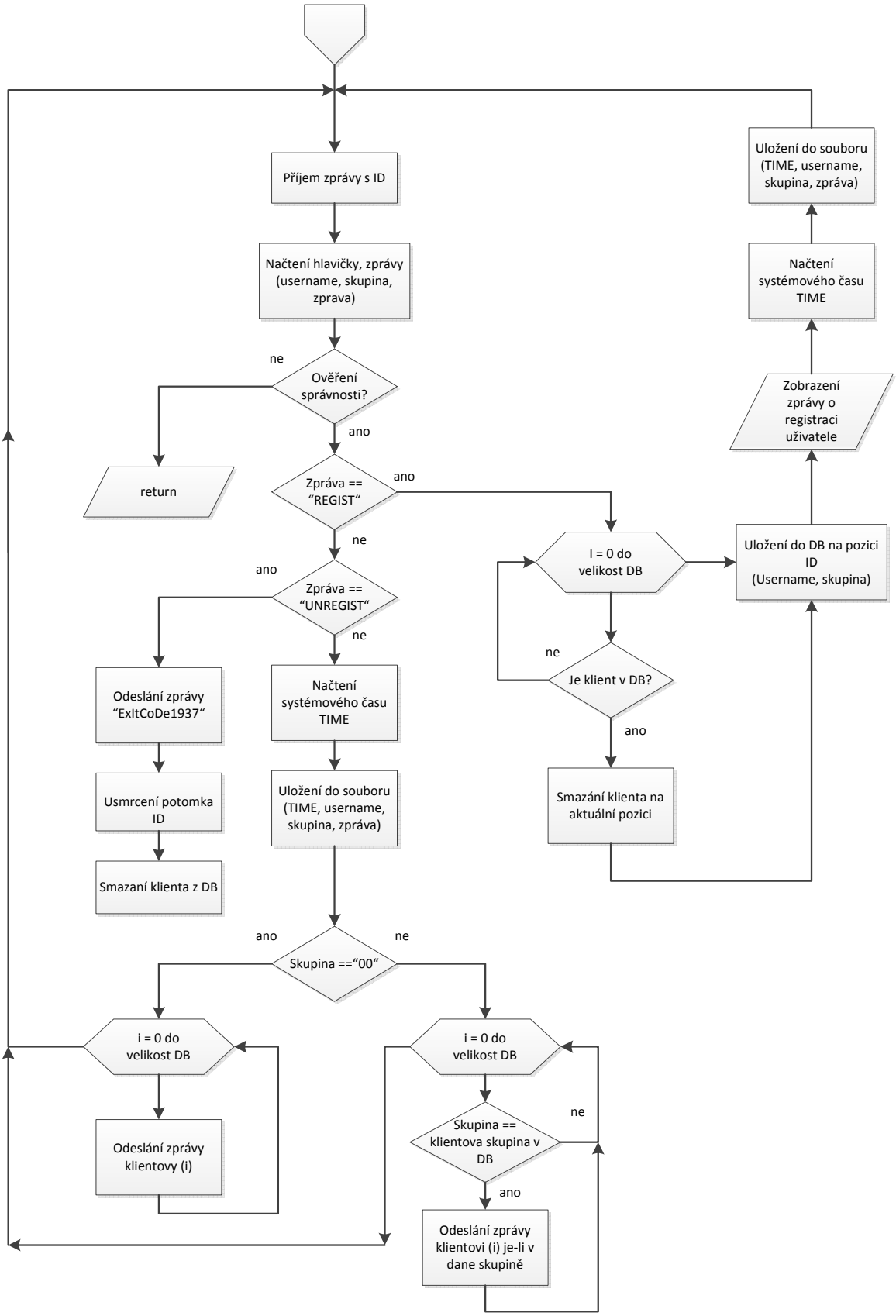
A Vývojové diagramy

B Manuál

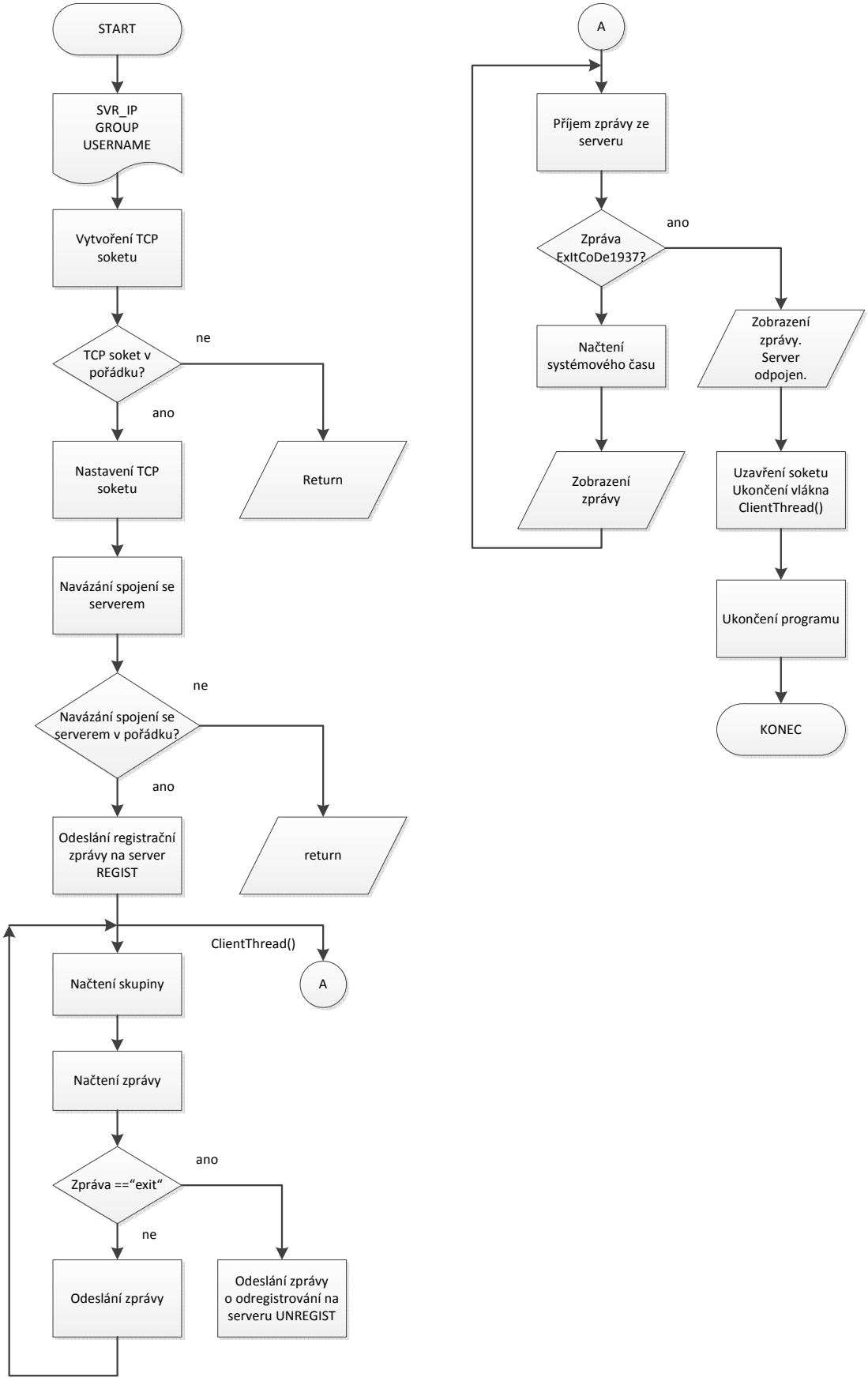
A. PŘÍLOHA: VÝVOJOVÉ DIAGRAMY



Obrázek A.1: Vývojový diagram serveru (1. část)



Obrázek A.2: Vývojový diagram serveru (2. část)

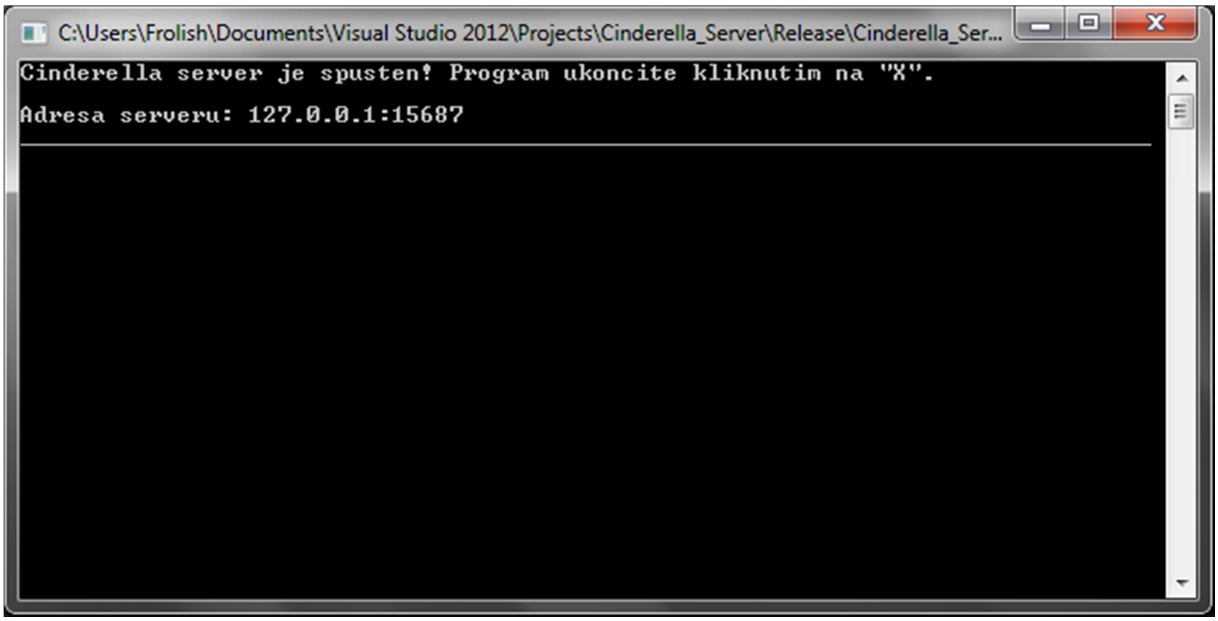


Obrázek A.3: Vývojový diagram klienta

B. PŘÍLOHA: MANUÁL

Program Cinderella je aplikace klient-server sloužící k zasílání krátkých zpráv v síti LAN. Pro správnou funkčnost programu je třeba mít na PC instalované komponenty run-time jazyka Visual C++ pro patřičnou architekturu x86 nebo x64, dostupné na stránkách společnosti Microsoft nebo na přiloženém DVD.

Serverovou aplikaci můžeme nainstalovat na jednu libovolnou klientskou stanici v síti. V souboru „settings.txt“ nastavíme IP adresu síťového adaptéru připojeného do sítě. Server spustíme „Cinderella_Server.exe“ a je-li vše správně nastaveno zobrazí se nám okno aplikace viz Obrázek B.4.



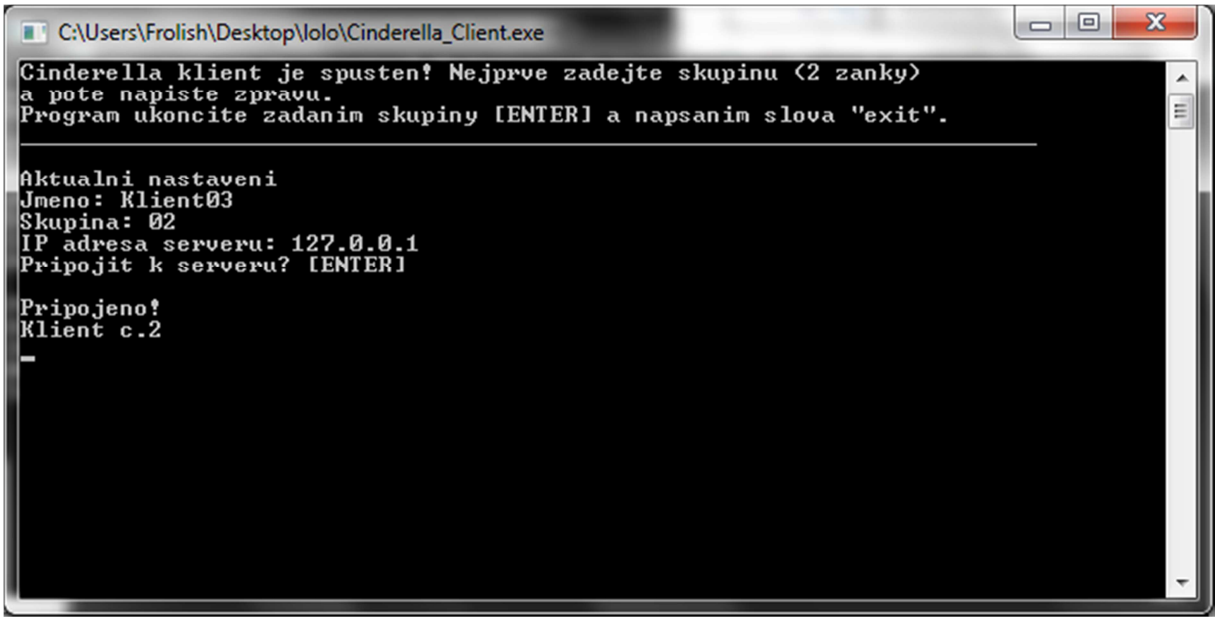
Obrázek B.4: Spuštěná aplikace Cinderella server

Klientskou aplikaci instalujeme na kteroukoli stanici v síti. Do souboru „settings.txt“ uložíme nastavení podle následujícího.

1. Řádek – IP adresa serveru (např.: 192.168.0.1)
2. Řádek – skupina (např.: 25)
3. Řádek – Uživatelské jméno (např.: Klient11)

Máme-li nastaveno, spustíme program a potvrdíme připojení k serveru stisknutím klávesy ENTER. Po úspěšném připojení můžeme zasílat zprávy zadáním skupiny [ENTER] Text zprávy [ENTER].

Běžící klientská aplikace je zobrazena na obrázku Obrázek B.5.



Obrázek B.5: Aplikace Cinderella klient